M T W T F S S

COMPASS

Date :

# Module 1: Language fondametals

* Introduction
* keywords
* Datatypes & Variables
* Operators
* Control Methods
* Identifiers
* Scanners
* arrays
* Strings

# Module 2: Object - Oriented Programming

* Introduction
* Static and non-Static
* class and Object
* Memory Management
* Intialization Blocks
* Constructors
* Inheritance
* Constructor Chaining
* Typecasting
* Method Overloading
* Method Overriding
* Polymorphism
* Abstraction
* Encapsulation
* Has-A Relationship

M T W T F S S

COMPASS

Date :

# Module 3: Core Java Libraries

* Introduction
* Class Object
* Exception Handling
* Class String
* String Buffer & String Builder
* Wrapper class
* File Handling
* Threads
* Collection Frameworks

M T W T F S S

COMPASS

Date : 18/03/25

* Program :- Program is a set of Instruction.

* Syntax : rules & regulations to be followed when writing programs.

## BASIC SYNTAX

The basic syntax for writting a java program consist of 2 parts.

1. Class Name
2. Main Methods

1. class : It is most basic fundamental unit of writing java program.

2. Main Method : It is entry point for execution inside a java program.

Syntax

```
class Classname
{
    public static void main (String[] args)
    {

    }
}
```

Note

() → Parenthesis
{} → Braces
[] → Brackets

M T W T F S S

Date :

Program 1

```
Class FirstJavaProgram
{
    public static void main (String[] args)
    {
        System.out.println ("Welcome to Java");
    }
}
```

O/P :- Welcome to Java

Program 2 : Write a program to display the name of your best friend.

```
class FriendName
{
    public static void main (String[] args)
    {
        System.out.println ("Riya Jain");
    }
}
```

O/P :- Riya Jain

* System.out.println ();

↳ It is a predefine java statement.

↳ It can be used by the programmers to display values in the output screen.

M T W T F S S

COMPASS

Date :

* String type values should be endosed in double qouts (" ").
* Character values should be enclosed in singal qouts (' ');
* Numbers & Booten value can be display directly.

# program 3

```
class Program1
{
    public static void main (String[] args)
    {
        System.out.println("Qspiders  Jspiders");

        System.out.println(143);

        System.out.println(7.45);

        System.out.println('J');

        System.out.println(true);
        System.out.println(false);
    }
}
```

O/P

```
Qspiders  Jspiders
143
7.45
J
true
false
```

M T W T F S S

COMPASS

Date :

* Alphanumaric

Brayan Write a java program to display the following details

1. Full Name
2. Year of Passing
3. CGPA
4. Grender (M/F)
5. Single (true/false)

```
class DisplayDetails
{
    public static void main(String[] args)
    {
        System.out.println("Shrasti Pawar");
        System.out.println(2024);
        System.out.println(8.42);
        System.out.println('F');
        System.out.println(true);
    }
}
```

Output :- Shrasti Pawar
2024
8.42
F
true

M T W T F S S

COMPASS

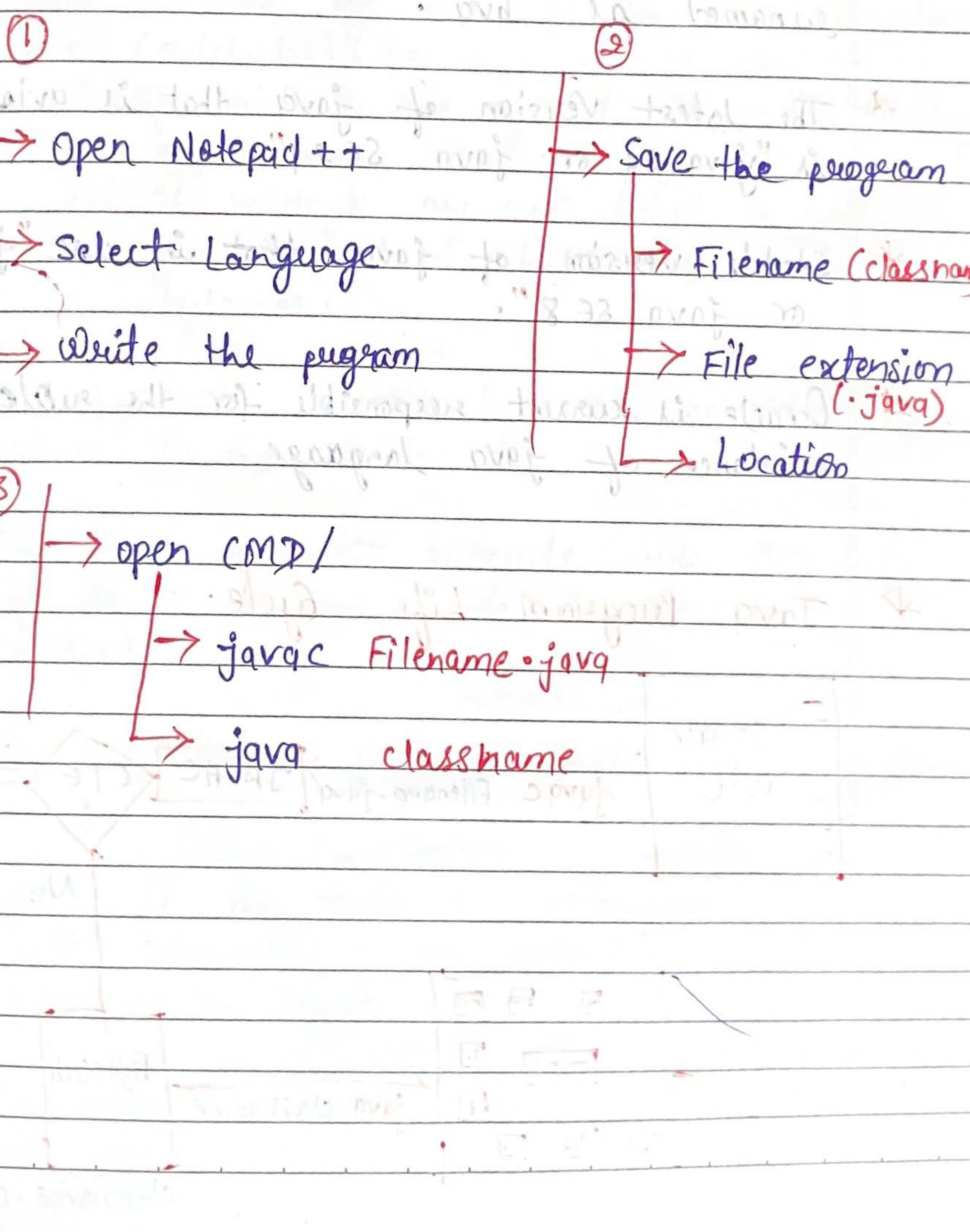
Date : 19/03/25

* DOWNLOAD

1) JDK (Java Development kit)
java.oracle.com

2) Notepad++

# Practical execution



M T W T F S S

Date :

## * History of java

* Java was first release in the year 1985 by "Sun Microsystems."

* James Gosling is the considered as a father of java programming language.

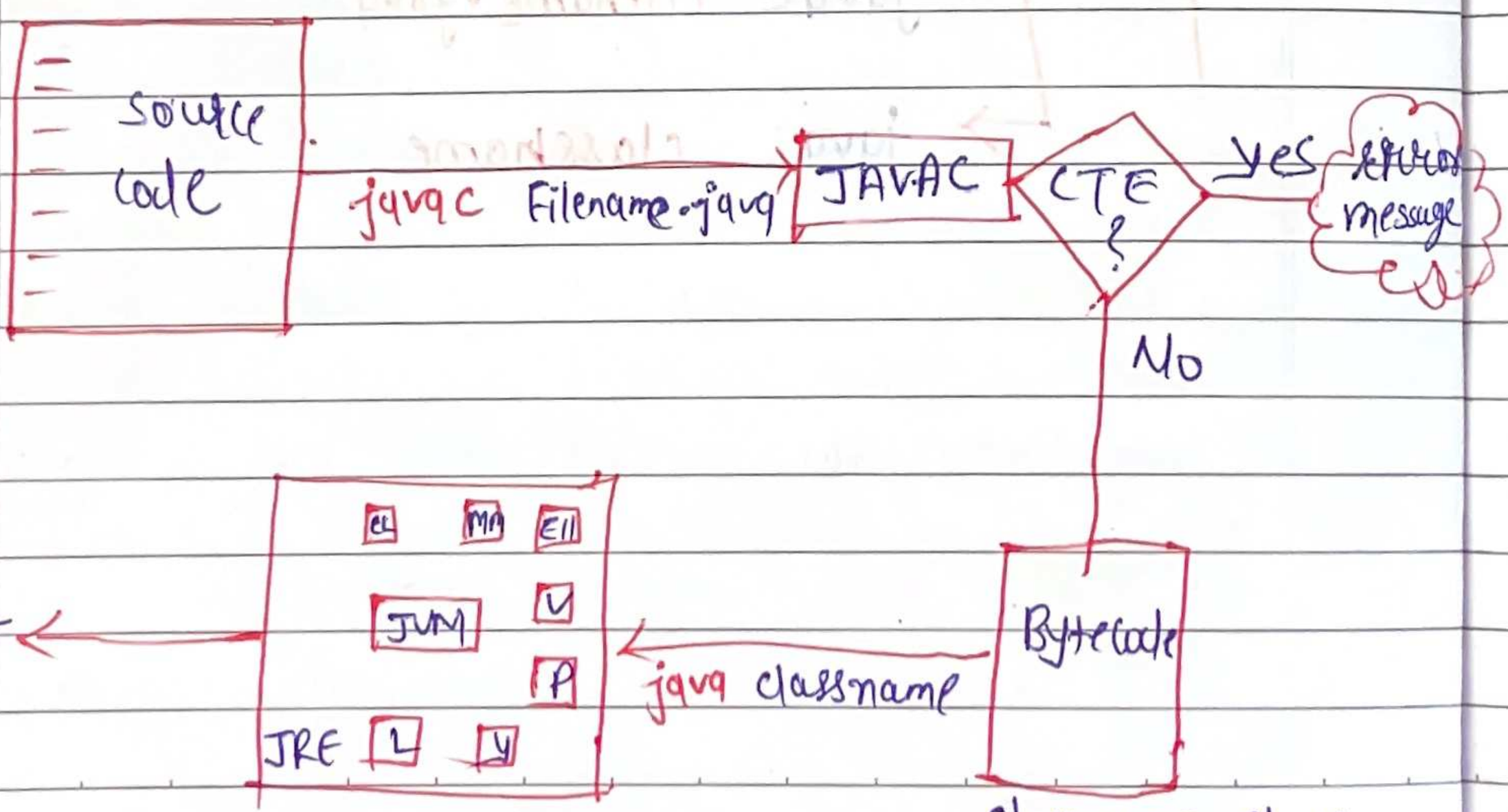
* Initily it was named "OAK" and letter renamed as "Java".

* The latest version of java that is available is "java 24 or java SE 24".

* Stable version of java ~~that~~ in use "java 8 or java SE 8".

* Oracle is current responsible for the rules and maintance of java language

## * Java Program Life Cycle.



M T W T F S S

COMPASS

Date : 20/03/25

(Source code) (Byte code)

HLC ——compiler——> ILC ——EE——> 0, 1

* The programer writes source code and save it by providing a filename and ".java" file extension.

* The ".java" file containing source code is given to JAVAC, The java compiler checks the program line by line to find compile time error (mistacks).

(i) If mistack are found it will display appropriate error message.

(ii) If mistack are not found it will generate ".class" file which contains bytecode.

* The bytecode generated by the compiler is given to the JVM.

* The JVM executes bytecode with the help of its subsystems inside the JRE to provide output.

* The programmer performs basic output verificatio

* Javac - (Java Compiler)
javac is the name of our java compiler. javac will take the source code present in .java file as an input and check for error.

→ If error is found it will display error message.

→ if there is no error it will generate .class file which contains bytecode

* → Bytecode : The Bytecode is the intermediate level code in java language.

→ It is created by the compiler and it is present inside the .class file.

→ The bytecode is not human readable.

→ Jvm makes use of bytecode for execution hence it is also called as java executable code.

* JVM (Java virtual machine) :-

→ The jvm is the execution engine of a java language.

→ It is responsible for executing the bytecode and providing output.

→ The jvm internally makes use of several subsystems. during execution.

→ It is a virtual machine and doesnot have a physical existance.

* JRE (Java Runtime Environment) :-

→ during program execution the jvm internally makes use of several books.

M T W T F S S

COMPASS

Date :

(i) Loader
(ii) Linker
(iii) Memory manager
(iv) Event Handler
etc.....

→ This collection of rules is represented as the JRE.

→ All activities inside the JRE is controlled by the JVM.

*→ JIT (Just In Time):-

→ It is a subsystem of JVM present inside the JRE.

→ It is responsible for converting bytecode to Machine level code.

* The "+" operator :-
In java using the ('+) operator we can perform the following two operatations
1. Addition
2. Concatenation (joining)

Note In java, Every character value is internally stored ~~used~~ using a number derived from UNICODE system

'A' ⟶ 'Z'
65 90

M T W T F S S

Date : 21/03/25

'a' → 'z'

97 122

'0' → '9'

1 1

48 57

* Order of Execution: In java the default order of execution is top to bottom & left to Right.

# class Example1
{
public static void main (String[] args)
{
System.out.println (12+13);
System.out.println (2.3 + 3.2);
System.out.println (10+2.5);
System.out.println (3.2+10);
}
}

# class Example2
{
public static void main (String [] args)
{
System.out.println ("Qspiders" + "Jspider");
System.out.println ("Boys" +90);
System.out.println (143 + "students");
}
}

M T W T F S S

COMPASS

Date :

output :-

Qspider Jspider
Boys 90
143 Students

# Comments or Note :- Comments or notes given by the programmer within the source code.
→ the programmer makes use of comments to discribe or explain code functionality.
→ Presence of comments make the program more easy to read and understand.
→ Comments not have any impact on program execution
→ In java comments are of two types.
(i) Single line comment (//)
(ii) Multi line comment (/* */)

# class Example3
{
public static void main (String[] args)
{
System.out.println (10+10)
System.out.println ("10" + "10");
System.out.println ("10" +10);
System.out.println (10+"10");
System.out.println ("A" +10);
System.out.println ('A' +10);
}
}

M T W T F S S

COMPASS

Date :

O/P:

20

1010

1010

1010

A10

75

# class Example 4
{
public static void main (String[] args)
{
System.out.println ("Qspiders" + " Testing");
System.out.println ("Jspiders" + " Java");
System.out.println ("Pyspiders" + " Python");
}
}

Output : Qspiders Testing
Jspiders Java
Pyspiders Python

# class Example5
{
~~plus~~ public static void main (String[] args)
{
System.out.println (20+20);
System.out.println (20+20+" is the number");
System.out.println ("The number is "+20+20);
System.out.println ("The number is "+(20+20));
}
}

M T W T F S S

Date : 22/03/25

Output: 40
40 is the number
The number is 2020
The number is 40

## keyWords

* keywords are predefine coords that have special meaning.
* As a programmer we can use keywords but we can not change the meaning.
* keywords are comptelly in lower case.
* generaly keyword appear in shade of blue colo
* java provides a total of 53 keywords.

byte, short, int, long, float, double, cha
boolen, void, class, enum, interface, extends,
implements, true, false, null, if, else,
switch, case, break, default, for, do, while,
assert, continue, static, final, abstract,
Synchronizatido, strictp, transient, native,
volatile, public, private, protected, package
import, try, catch, finally, itheows, throw
goto, const

Note

goto & const keyword are not used directly
in java programming.

M T W T F S S

Date :

# DATATYPES & VARIABLES

* Datatypes and variables can be used by the programmer to store values inside a java program.

* **Data Types :-** Datatypes are tokens of java programming that helps to identify what type of value will be store inside variable.

→ Datatypes in java are classified into two types.

(i) primitive datatypes
(ii) non-primitive datatypes

* **Primitives :** Primitives are inbuild datatypes.
→ Primitives are used to declare variables.
→ Java provides total of 8 primitives.

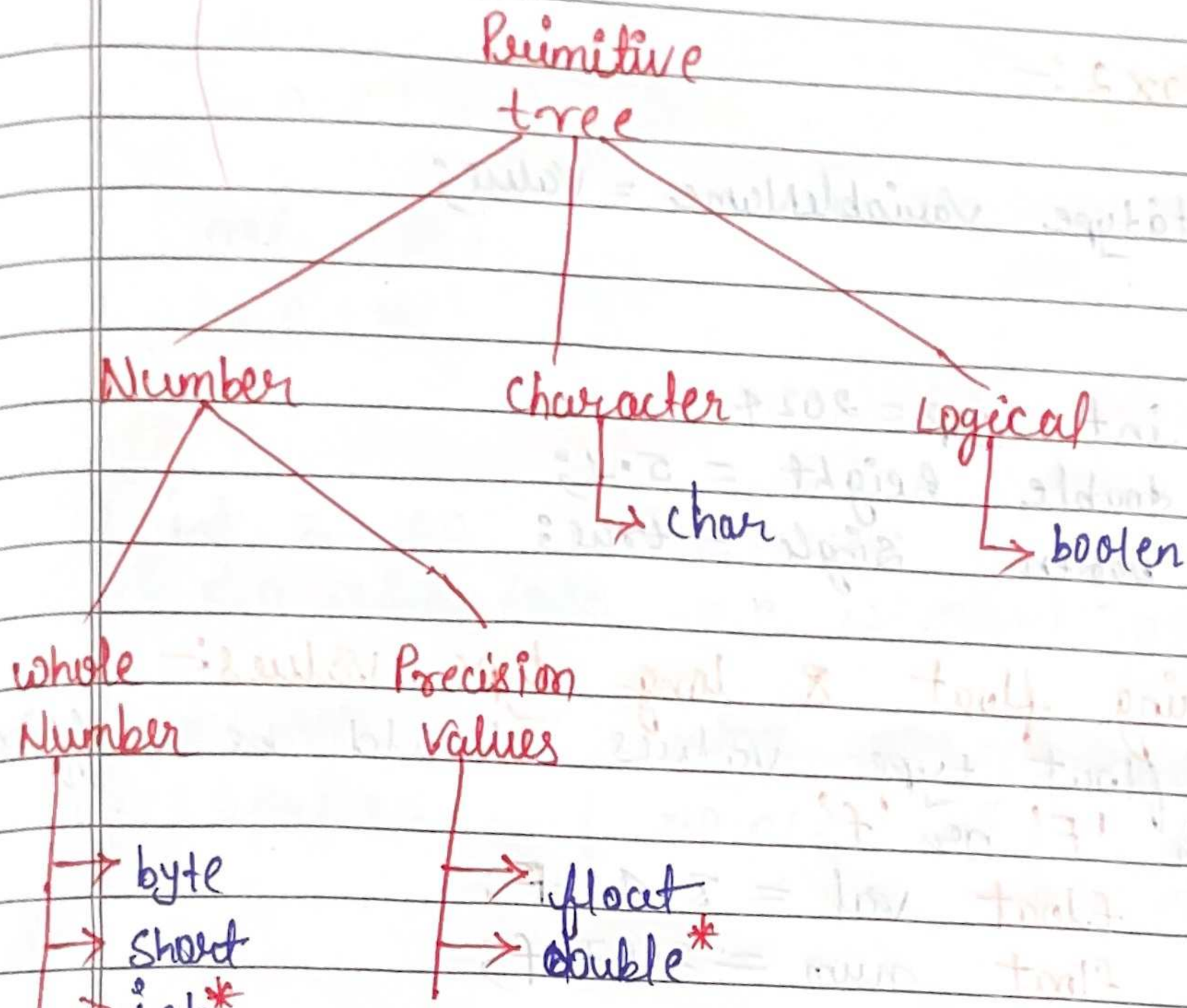
byte ⟶ 1-Bytes
short ⟶ 2-Bytes
int ⟶ 4-Bytes
long ⟶ 8-Bytes
float ⟶ 4-Bytes
double ⟶ 8-Bytes
char ⟶ 2-Bytes
boolean ⟶ 1-Bit

8-Bit ⟶ 1 Byte

M T W T F S S

COMPASS

Date :




☆ Variables :- Variables are container used to store values.

⇒ The variable name is always decided by the programmer.

→ Syntax for creating the variable

⇒ Syntax 1 :

datatype variablename; → Declaration
variablename = value; → Initialization

Ex :-

① int age;
age = 22;

② double cgpa;
cgpa = 8.42;

③ char ch;
ch = 'J';

M T W T F S S
☐ ☐ ☐ ☐ ☐ ☐ ☑

COMPASS
Date : 23/03/25

Syntax 2:-

datatype variableName = value;

Ex:

int yop = 2024
double height = 5.1;
boolen sigle = true;

* Storing float & long type values:-
→ All float type values should be suffixed with 'F' or 'f'

float val = 3.42-F;
float num = 2.67 f;

→ All long type values should be suffixed with 'L' or 'l'.

long aadhar = 123412341234 L;
long creaditCardNumber = 4678234523l l;

* Re-Initialization
Replacing old value in a variable with new value is called as reinitialization

Ex:- int q
q = 10; → Initialization
=
=
=
q = 20; → Re-initialization

→ Example: int ml
class ml = 30;
S.O.P (ml);

M T W T F S S

COMPASS

Date :

```
ml = 60;
s.o.p(ml)

ml = 90;
s.o.p(ml)
```

Ex

```
int x = 100
s.o.println(x);

x = 200
s.o.p(x);
```

```
int x = 100;
s.o.p(x)

int x = 200
s.o.p(x)
```

→ CTE

```
int x = 100
s.o.p(x)

float x = 7.4
```

→ CTE

Redeclaration is not allowed

# FINAL VARIABLES

→ Any variable that is declare with 'FINAL' key word is called as final variable.

```
final int    a = 10;
final double b = 12.5;
final char   c = 'J';
```

Note If a variable is declare with 'final' keyword. it can not be reinitialized

→ We can use final variable to create constant values in a java program.

Ex

| Without final keyword | with final keyword |
| --- | --- |
| int x = 10 | final int x = 10; |
| S.O.P(x); | S.O.P(x); |
| x = 2 | x = 20 //CTE |
| S.O.P(x) | |

## * Data Members and Local Variables :-

→ In java a programmer can declare variables either inside a method or directly in the class body.

→ If variables are declared directly in the class body they are called Data members.

→ If variables are declared inside a method they are called local variables.

```
Ex: class example
    {
        int    a=10;
        double b=3.4;        ---- DATA MEMBERS
        char   ch='J';

        public static void main (String[] args)
        {
            int x=20;
            double y=5.6;    ---- LOCAL VARIABLES
            char z='Q';
        }
    }
```

M T W T F S S

COMPASS

Date :

# OPERATORS : Operators are symbols that performs perticular task.

→ In Java Operators are classifed into following familys.

1. Arithmatic Operators: (+, -, *, /, %, =)
2. Relational Operators : (<, >, <=, >=, ==, !=)
3. Logical Operators : (&&, ||, !)
4. Bitwise Operator : (&, |, ^, <<, >>)
5. Unary Operator : (++, --)

## * Arithmatic Operators

| | | |
|---|---|---|
| + | ——— | Addition |
| - | ——— | substraction |
| * | ——— | Multiplication |
| / | ——— | Division |
| % | ——— | Modulus |
| = | ——— | Assignment |

```
class Program1 :
{
    public static void main (String[] args)
    {
        int a = 10
        int b = 2;
```

M T W T F S S

COMPASS

Date :

* Bigger datatype :-

→ Ability to store precision value
→ Memory size

byte → short → int → long → float → double
char → int

→ (int / int) → int
(double / double) → double
(int / double) → double
(double / int) → double

(int / long) → long
(float / double) → double
(long / float) → float

```
class Program2
{
    public static void main (String[] args)
    {
        int p = 7;
        int q = 2;
        System.out.println (p/q);

        double x = 7.2;
        double x = 2.0;
        System.out.println (x/y);
        System.out.println (p/y);
        System.out.println (x/q);
    }
}
```

M T W T F S S

COMPASS

Date :

* Operator Precedence

[*, /, %] → 1st

[+, -] → 2nd

[=] → 3rd

```
class Program3
{
    public static void main(String[] args)
    {
        System.out.println(10+2*4+3);
        System.out.println(10*2+4*3);
        System.out.println(10*2/4+3);
        System.out.println(10/2*4+3);
        System.out.println(10%2+4*3);
        System.out.println(10-2%4*3);
        System.out.println(10*2-4%3);
    }
}
```

# Square of n

```
class Program4
{
    public static void main(String[] arg)
    {
        int n=10;
        int square = n*n;
        System.out.println(n+" Square is : "+
                           square);
    }
}
```

M T W T F S S

COMPASS

Date :

## cube of n

```
class Program5
{
  public static void main (String [] args)
    {
      int n=10;
      int cube = n*n*n;
      System.out.println (n+" cube is : "+cube);
    }
}
```

## Given radius and pi

```
class Program6
{
  public static void main (String [] args)
    {
      double radius = 4.5;
      double pi = 3.14;
      double area = pi * radius * radius;
      double circumference = 2* pi * radius;
      System.out.println("Area of circle : "+area);
      System.out.println(" Circumference of circle:"
                                    +circumference);
    }
}
```

₹50,000
100

M T W T F S S

COMPASS

Date :

# Write a java program to calculate Simple area & perimeter of square.

```
class Program7
{
    public static void main(String [] args)
    { double side = 10.0;
      double area = side * side;
      double perimeter = 4 * side;

      System.out.println("Area of square :" + area);
      System.out.println("Perimeter of square :" + perimeter);
    }
}
```

# Write a java program to calculate simple interest using the data given below.

principle = 100000.0
rate = 12.5
tenure = 2 years
SimpleInterest = ?

⇒

```
class Program8
{
    public static void main(String[] args)
    {
        double principle = 100000.0;
        double rate = 12.5;
        int tenure = 2;
        double simpleInterest = (principle * tenure * rate)/100;
        System.out.println(simpleInterest);
    }
}
```

M T W T F S S

Date : 25/03/24

## * Relational Operator

1. < ⟶ Less than
2. > ⟶ Greater than
3. <= ⟶ Less than equal to
4. >= ⟶ greater than equal to
5. == ⟶ Equal to
6. != ⟶ Not equal

Relational operatores are used for comparision after comparision it return true or false.

```
# class Program1
{
   public static void main (String[] args)
   {
      System.out.println (12<15); // true
      System.out.println (17<15); // false

      System.out.println (13>14); // true
      System.out.println (10>10); // false
   }
}
```

```
# class Program2
{
   public static void main (String[] args)
   {
      System.out.println (13<=16); // true
                         (19<=16); // false
                         (16<=16); // true
```

M T W T F S S

COMPASS

Date :

(14>=12); // true
(11>=12); // false
(12>=12); // true

}
}

# class Program 3
{
public static void main(String[] args)
{
System.out.println(143 == 420);
——— // ——— (90 == 90);
——— // ——— (15 != 15);
——— // ——— (20 != 15);
}
}

# class Program 4
{
public static void main (String[] args)
{
System.out.println(10<10.0) // false
(10>10.0) // false
(10==10.0) // true
('A'=='a') // false
('A'==65) // true
('A'=="A") // CTE
}
}

M T W T F S S

COMPASS

Date :

## * Logical Operators

1 && ⟶ Logical AND
2 || ⟶ Logical OR
3 ! ⟶ Logical NOT

→ logical operators are used to bilit complex condition that involve multiple comparision.

→ The logical operators will compare booten values and provide booten values. as result.

## * Decimal ⟶ Binary

25

| | | |
|---|---|---|
| 2 | 25 | |
| 2 | 12 | 1 |
| 2 | 6 | 0 |
| 2 | 3 | 0 |
| | 1 | 1 |

25 ⇒ 1 1 0 0 1

## * Binary to Decimal

1 1 0 0 1 0 1 0
* * * * * * * *
$2^7$ $2^6$ $2^5$ $2^4$ $2^3$ $2^2$ $2^1$ $2^0$
128 64 32 16 8 4 2 1

128 + 64 + 0 + 0 + 8 + 0 + 2 + 0

⇒ 202

M T W T F S S

COMPASS

Date :

* 1100 11 → 51

1 1 0 0 1 1

* * * * * *

$2^5$ $2^4$ $2^3$ $2^2$ $2^1$ $2^0$

32 + 16 + 8 + 4 + 2 1

32 + 16 + 0 + 0 + 2 + 1

⇒ 51

## * Bitwise Operators :

1. & ⟶ Bitwise and
2. | ⟶ Bitwise OR
3. ^ ⟶ Bitwise XOR
4. << ⟶ Bitwise Left shift
5. >> ⟶ Bitwise Right shift

| Bitwise AND (&) | | |
|---|---|---|
| INPUT 1 | INPUT 2 | OUTPUT |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| Bitwise OR (1) | | |
|---|---|---|
| INPUT 1 | INPUT 2 | OUTPUT |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

M T W T F S S

Date :

| Bitwise XOR (^) | | |
|---|---|---|
| Input 1 | Input 2 | Output |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# int x = 25
int y = 21
S.O.P(x & y) // 17
S.O.P(x | y) // 29
S.O.P(x ^ y) // 12

25 → 11001 ^ 11001
21 → 10101 10101
11101
$2^4\ 2^3\ 2^2\ 2^1\ 2^0$
16 + 8 + 4 + 1 = 29

11001
10101
01100
$2^3\ 2^2$
8 + 4 = 12

Q 10 → 1010
23 → 10111

and
01010 (1) 01010 ^ 01010
10111 10111 10111
00010 11111 11101
$2^1$ 16 8 4 2 1 16 8 4 0 1
⇒ 2 ⇒ 31 ⇒ 29

M T W T F S S

Date: 26/03/24

## ☆ Bitwise left shift operator (<<)

```
int c = 21
sop(c<<1); → 42
```

1 0 1 0 1 0

$2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

32 16 8 4 2 1

32 + 8 + 2 = 42

```
int a = 10
S.O.p(a<<1); → 20
```

1 0 1 0

1 0 1 0 0

$2^4\ 2^3\ 2^2\ 2^1\ 2^0$

16 8 4 2 1

16 + 4 = 20

```
int q = 9;
S.O.p(q<<2); → 86
```

```
int s = 5;
S.O.p(s<<3); 44
```

2 | 9

2 | 4 — 1

2 | 2 — 0

0

0 0 1

1 0 0 1 0

1 0 0 1 0 0

32 16 8 4 2 1

32 + 4 = 36

#

# class Program 5

{

public static void main (String [] args)

{

```
int p = 13;
S.O.p(p<<2); 1152
int q = 9;
S.O.p(q<<2); // 36

int r = 25;
S.O.p(r<<3); // 400
}
}
```

1 0 1

1 0 1 0

1 0 1 0 0

36 16 8 4 2 1

36 + 8 = 44

M T W T F S S

Date: 4/1

## ☆ Bitwise Right shift (>>)

```
int z = 19;
S.O.P (z >> 1); → 9
```

19 → 1 0 0 1 1

1 0 0 1 | 1 last
2³ 2² 2¹ 2⁰

8 4 2 1

8 + 0 + 0 + 1 = 9

* int y = 37

S.O.P (y >> 1); → 18

# class Program6
{
  public static void main (String [] args)
  {
    int x = 25;
    System.out.println (x >> 1); // 12

    int y = 37;
    System.out.println (y >> 1); // 18
  }
}

int q = 50;
S.O.P (q >> 2); → 12

50 → 1 1 0 0 1 0

1 1 0 0 1 | 0 last
1 1 0 0 | 1 last

8 4 2 1

8 + 4 ⇒ 12

int r = 84;
S.O.P (r >> 3); // 10

84 → 1 0 1 0 1 0 0

1 0 1 0 1 0 | 0
1 0 1 0 1 | 0
1 0 1 0 | 1

8 4 2 1

8 + 2 = 10

int s = 78;
S.O.P (s >> 3); → // 9

78 → 1 0 0 1 1 0 0

1 0 0 1 1 0 | 0
1 0 0 1 1 | 0
1 0 0 1 | 1

8 4 2 1

8 + 1 = 9

M T W T F S S

COMPASS

Date :

★ use of bitwise operator
(1) embedaded software
(2) cyber security

## ✡ Unary Operator:

In java we have the following unary operators.
1. ++ (Increment)
2. -- (Decrement)

→ The increment operator (++) will increase the value is a ~~zero~~ variable by 1.

→ The decrement operator (--) will decrease the value in a variable by 1.

→ The unary operations increment and decrement can be done in two ways.
(1) post operation
(2) pre operation

Some question

Q
```
int a = 27;  ⟶  1 1 0 1 1
int b = 25;  ⟶  1 1 0 0 1
int c = a|b; ⟶  1 1 0 1 1
int b = a^b; ⟶  0 0 0 1 0
int e = c&d; ⟶  0 0 0 1 0
s.o.p(e) // 2        2^1 2^0
                     ⇒ 2
```

Q
```
int p = 45;  ⟶  1 0 1 1 0 1
int q = 9;   ⟶  0 0 1 0 0 1
int r = p>>1 ⟶  0 1 0 1 1 0
int s = q<<1 ⟶  0 1 0 0 1 0
int t = r^t  ⟶  0 0 0 1 0 0 ⟶ 4
sop(t); // 4
```

16 8 4 2 1

M T W T F S S

COMPASS

Date : 27/01/25

Q. int a = 27; → 1 1 0 1 1
int b = 26; → 1 1 0 1 0
int c = a & b; → 1 1 0 1 0
int d = a | b; 1 1 0 1 1
int e = c ^ d; 0 0 0 0 1
s.o.p(e); 1 → 1

★ POST OPERATOR

syntax | Variable ++ / variable --

* int y = 17; // 16, 15
s.o.p (y--); // 17
s.o.p (y--); // 16
s.o.p (y); // 15

int x = 13; // 14, 15
s.o.p (x++); // 13
s.o.p (x++); // 14
s.o.p (x); // 15

* int p = 10; // 11
int q = p++ (+) q++;
10 11

int r = 15; // 14
int s = r-- (+) r--;
15 14

s.o.p (q); // 21

s.o.p (s); 29

* int a = 7; // 8, 9
int b = a++ (+) a++ (+) a++;
7 8 9

int c = 5; // 4, 3
int d = c-- (+) c-- (+) c--;
5 4 3

s.o.p (b) // 24

s.o.p (d); // 12

* int x = 13; // 14, 15
int y = 18; // 17, 16
int z = x++ (+) y-- (+) x++ (+) y--;
13 18 14 17
s.o.p (z); // 62

* int p = 15; // 14, 13
int q = 12; // 13, 14
int r = p-- (+) q++ (+) p-- (+) q++;
15 12 14 13
s.o.p (r); // 54

M T W T F S S

COMPASS

Date :

☆ Pre Operation :- In case of pre operation first increment or decrement will be done. and then value will be use, to.

Syntax

++ Variable / -- Variable

* int a = 5;
S.O.P (++a); //6
S.O.P (a); //6

* int b = 8;
~~int~~ S.O.P (--b); //7
S.O.P (b); //7

* int x = 11;
S.O.P (++x); //12
S.O.P (++x); //13
S.O.P (x); //13

int y = 17;
~~int~~ S.O.P (--y); //16
S.O.P (--y); //15
S.O.P (y); //15

* int a = 14; //15, 16
int b = ++a (15) ⊕ ++a (16);
S.O.P (b); //31

int c = 19; //18, 17
int d = --c (18) ⊕ --c (17);
S.O.P (d) //35

* int p = 12; //13, 14
int q = ++p (13) ⊕ ++p (14) ⊕ ++p (15);
S.O.P (q) //42

int r = 18 //17, 16
int s = --r (17) ⊕ --r (16) ⊕ --r (15);
S.O.P (s); //48

* int a = 12; //13, 14
int b = 15; // 14, 13
int c = ++a (13) ⊕ --b (14) ⊕ ++a (14) ⊕ --b (13);
S.O.P (c); //54

int x = 16; //15, 14
int y = 14; //15, 16
int z = --x (15) ⊕ ++y (15) ⊕ --x (14) ⊕ ++y (16);
S.O.P (z); //60

M T W T F S S

COMPASS

Date :

* int a = 7; // 8, 9.
int b = a++ + ++a + a++;
7 9
S.O.P(b) // 16

* int c = 5 // 4, 3
int d = --c + c--;
// 4 4

S.O.P(d); // 8

* int q = 10; // 11, 10
int b = 15; // 14, 15
int c = q++ + --b + a-- + ++b;
10 14 11 15

S.O.P(c); // 50

* int x = 13; // 12, 13.
int y = 18 // 19, 18
int z = --x + y++ + x++ + --y;
12 18 12 18
S.O.P(z); // 60
S.O.P(y); // 18

* double val = 4.5;
S.O.P(++val); // 5.5

* int num = 0;
S.O.P(--num); // -1

Very Imp question for interview

char ch = 'a';
S.O.P(++ch); // b
(a) 98
(b) A
(c) b ✓
(d) CTE

* char ch = 'a';
S.O.P(++c); → b
S.O.P(1+ch); → 98

M T W T F S S

COMPASS

Date :

* boolean b = true;
S.O.P (++b);
(a) CTE ✓ (b) true (c) false (d) uv true

* S.O.P (++7);
(a) 8
(b) 7
(c) CTE ✓
(d) Junk

Note
# Java not generate junk value.

M T W T F S S

COMPASS

Date : 28/03/25

# CONTROL STATEMENTS

* In java we can use control statements to alter the order of execution inside a java program.

* In java we have following control statements

~~Switch Statement~~

1. If - Statement
2. Switch - Statement
3. For - ~~Statement~~ loop
4. While - ~~Statement~~ loop

Q.1 Write a program to swap given two no.

```
class Program 1
{
    public static void main (String [] args)
    {
        int a=10;
        int b =20; int c;
        c = a;
        a = b;
        b = c;
        System.out.println(a);
        System.out.println(b);
    }
}
```

M T W T F S S

10/10

COMPASS

Date :

Q write a program to swap two no. with using temporary variable.

→

```
class program1
{
public static void main (String [] args).
{
    int a=10;
    int b=20;
    a=a+b;
    b=a-b;
    a= a-b;
    System.out.println (a);
    System.out.println (b);
}
}
```

a = a+b // 30
b = a-b  30-20 // 10
a = a-b  30-10 // 20

# Assignment operator : Assignment operator are used to assign the value into a variable.

Variable = Value

* Compound Assignement operator
(+=, -=, *=, /=, %=)

* int a=10;
int b=20;
a+=b; // a=a+b
S.O.P(a); // 30
S.O.P(b); // 20

* int a=20
int b=10
a-=b; // a=a-b
S.O.P(a); // 10
S.O.P(b); // 10

M T W T F S S

Date :

```
* int a = 10;
  int b = 20;
  int c = 30
  a += b+c; //a=a+b+c
  S.O.P(a); //60
  S.O.P(b); //20
  S.O.P(c); //30
```

```
* int a = 10
  int b = 20
  int c = 30
  c=a+b;
```

Note: In which variable the result store that variable should be one of the operand.

## * If - Statements

* Versions:- Simple If statement

Syntax

```
if (condition)  --> true
                --> false
{
   // if - body
}
```

Ex

```
class Program1
{
   public static void main (String[] args)
   {
      System.out.println ("Program Starts");
      if (false)
      {
         system.out.println ("Java Full stack");
      }
      System.out.println ("Program End");
   }
}
```

M T W T F S S

Date : 02/04/25

## * Version 2 : If-else statement

Syntax

```
if(conditions)
{
    // If body
}
else
{
    // else body
}
```

# For class Programs

```
{
    P - -
    {
        if (true)
        {
            S.O.P("Developer");
        }
        else
        {
            S.O.P ("Tester");
        }
    }
}
```

M T W T F S S

Date :

```
# int n = 15;
  if (n % 2 == 0)
  {
    S.O.P("Even Number");
  }
  else
  {
    S.O.P("odd Number");
  }
}

# int age =
  if (age>=18)
  {
    S.O.P("Major");
  }
  else
  {
    S.O.P("Minor");
  }

# double accBal = 10000.0;
  int amt = 3000;
  if (amt <= accBal)
  {
    S.O.P("Withdrawal success");
  }
  else
  {
    S.O.P("Insufficent Balance");
  }
```

M T W T F S S

COMPASS

Date :

* Version 3 : If - else - If statement / Ladder.

Syntax

```
if (condition -1)
{

}
else if (condition-2
{

}
else if (condition 3)
{

}
else
{

}
```

# 
```
int n=13;
if (n%2==0 && n%3==0)
{
   S.O.P("Bangalore")
}
else if (n%2==0)
   {
     S.O.P("Chennai");
   }
else if (n%3==0)
  { S.O.P("Hydrabad");
  {
else
  {S.O.P("Cochin");
  {
```

M T W T F S S

Date :

```
# int age=
if (age <=3)
  {
    S.O.P ("Free");
  }
else if (age >3 && age<=10)
  {
    S.O.P ("Half");
  }
else if (age>10 && age<60)
  {
    S.O.P ("full");
  }
else
{ S.O.P ( "Sr. Citizen");
  }

# double per=
if (per >= 85)
  {
    S.O.P ("Distinction");
  }
else if (per >= 70 && per < 85)
  {
    S.O.P ("First class");
  }
else if (per >=60 && per<70)
  {
    S.O.P ("Second class");
  }
else if (per >=35 && per<60)
  {
    S.O.P ( "pass");
  }
else {S.O.P ("fail");
```

M T W T F S S

Date : 03/04/25

* Version 4 : NESTED - If statement

```
if ( )
{
  if ( )
  {
    if ( )
    {
    }
  }
}
```

# 
```
int a=5;
if (a%2==0)
 { if (a<10)
    { s.o.p("Masala Puri");
    }
  else
    { s.o.p ("Bhel Puri");
    }
 }
else
{ if (a<10)
    { s.o.p("pani puri");
    }
  else
    { s.o.p("sev puri");
    }
}
```

M T W T F S S

Date :

#
```
double accBal = 1000.0;
int amt = 13000;
if (amt <= accBal)
{
    if (amt % 100 == 0)
    {
        S.O.P ("withdrawl success");
    }
    else
    {
        S.O.P ("Invalid Denomination");
    }
}
else
{
    S.O.P ("Insufficient Balance");
}
}
```

#
```
char gender = 'M';
int age = 17;

if (gender == 'M')
{
    if (age >= 21)
    {
        S.O.P ("Eligible Bachelor");
    }
    else
    {
        S.O.P ("Boy Underage");
    }
}
else if (gender == 'F').
```

COMPASS

Date :

```
{ if (age>=18)
  {s.o.p("Eligible spinster");
  }
else
  {so.p("Girl Underage");
  }
}
else
{ s.o.p("select valid Gender");
}
```

* SWITCH - STATEMENT

syntax

```
switch(choice)
{
  case label 1;
  break;
  case label 2;
  break;
  case label 3;
  break;
  case label 4;
  break;
  :
  default
}
```

#

```
int choice = 5;
switch (choice)
{
  case 1: S.O.P("COD");
  break;
  case 2: S.O.P("CARD");
  break;
```

```
        case 3: S.O.p("UPI");
        break;
        case 4: SOP("Net Banking");
        break;
        default: S.O.P("Invalid choice");
    }
```

```
int option = 2;
swich (option)
 {
    case 1: S.O.p("English");
    break;
    case 2: S.O.p("kannada");
    break;
    case 3: S.O.p("Hindi");
    break;
    default: S: O.p("Invalid Selection")
  }
```

M T W T F S S

COMPASS

Date : 4/4/25

## * FOR - LOOP

Syntax

```
for (initialization ; condition ; operation)
                    (true)
{
  // for loop body
}
```

① ② ④ ③

# for (int a=1; a<=5; a++)

```
{
  S.O.P ("Jspider");
}
```

Jspider

# for (int x=5, x>0; x--)

```
{
  S.O.P ("Bangalore");
}
```

Bangalore

Tracing

| | | | |
|---|---|---|---|
| a=1 ✓ | Jspider | x=5 ✓ | Bangalore |
| a=2 ✓ | Jspider | x=4 ✓ | Bangalore |
| a=3 ✓ | Jspider | x=3 ✓ | Bangalore |
| a=4 ✓ | Jspider | x=2 ✓ | Bangalore |
| a=5 ✓ | Jspider | x=1 ✓ | Bangalore |
| a=6 ✗ | | x=0 ✗ | |

# for (int p=1; p<=10; p=p+2)

```
{
  S.O.P ("Developer");
}
```

| | |
|---|---|
| p=1 ✓ | Developer |
| p=3 ✓ | Developer |
| p=5 ✓ | Developer |
| p=7 ✓ | Developer |
| p=9 ✓ | Developer |
| p=11 ✗ | |

COMPASS

M T W T F S S

Date :

# for (int r=10; r>0; r=r-3)
{
S.O.p("Automation");
}

r=10 ✓ Automation
r=7 ✓ Automation
r=4 ✓ Automation
r=1 ✓ Automation
r=-2 ✗

# int n=5;
for (int i=0; i<n; i++)
{
S.O.p("*");
}

i=0
0<5 ✓ *
1<5 ✓ * *
2<5 ✓ * * *
3<5 ✓ * * * *
4<5 ✓ * * * * *
5<5 ✗

* System.out.println("Hello"); // move to next line
> Hello
> _

* System.out.print("Hello"); // same line
> Hello _

M T W T F S S

COMPASS

Date :

```
int n=5; // Size of Square
for (int i=0; i<5; i++)          // No of rows
{
    for(int j=0; j<n; j++)       // No of columns
    {
        S.O.println("*" + "  ");
    }
    S.O.println(); //Move To New Line.
}
```

n = 5

| | j=0 | j=1 | j=2 | j=3 | j=4 | j=5 |
|---|---|---|---|---|---|---|
| i=0 | * | * | * | * | * | |
| i=1 | * | * | * | * | * | |
| i=2 | * | * | * | * | * | |
| i=3 | * | * | * | * | * | |
| i=4 | * | * | * | * | * | |
| i=5 — | | | | | | |

M T W T F S S

Date : 05/04/25

n=5

| i=0 | * | * | * | * | * |
|---|---|---|---|---|---|
| i=1 | * | * | * | * | |
| i=2 | * | * | * | | |
| i=3 | * | * | | | |
| i=4 | * | | | | |

j=0 j=1 j=2 j=3 j=4

n=5

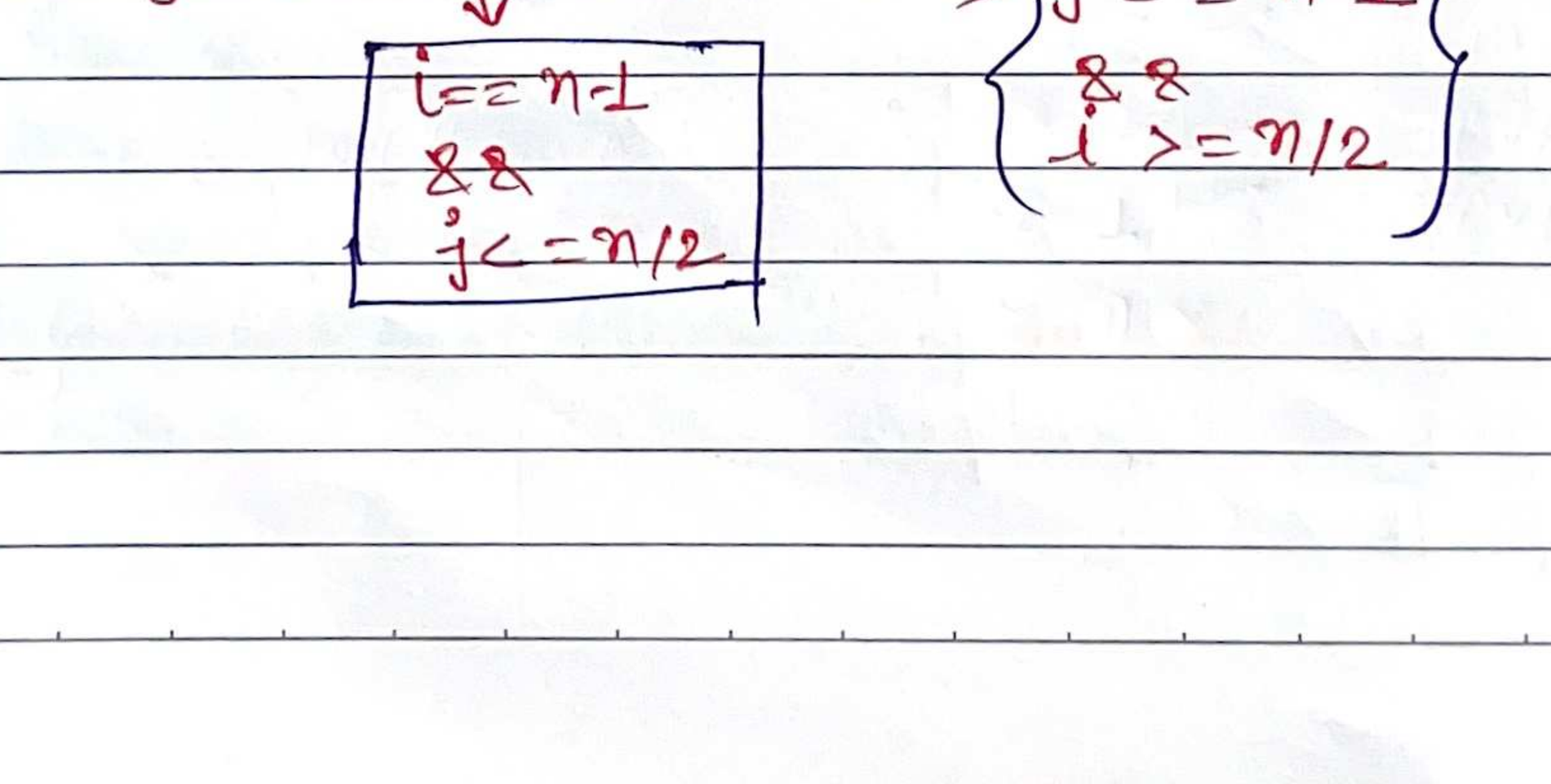
| i=0 | * | * | * | * | * → i+j==n-1 |
|---|---|---|---|---|---|
| i=1 | * | * | * | * | |
| i=2 | * | * | * | * | * → i==n/2 |
| i=3 | * | * | * | * | |
| i=4 | * | * | * | * | * → i==n-1 |

j=0 j=0 j=1 j=2 j=3 j=4

i==j

j=0 j=n/2 j==n-1

Q1



COMPASS

Date : 07/04/25

M T W T F S S

# While -LOOP

Syntax

```
while (condition)
{
  // while body
}
```

#

```
int a=1;
while (a<=5)
{
  S.O.println("Biryani");
  a++;
}
```

a=1 ✓ Biryani
a=2 ✓ Biryani
a=3 ✓ Biryani
a=4 ✓ Biryani
a=5 ✓ Biryani
a=6 ✗

#

```
int x=5;
while (x>0)
{
  S.O.pln("JAVA");
  x--;
}
```

x=5 ✓ JAVA
x=4 ✓ JAVA
x=3 ✓ JAVA
x=2 ✓ JAVA
x=1 ✓ JAVA
x=0 ✗

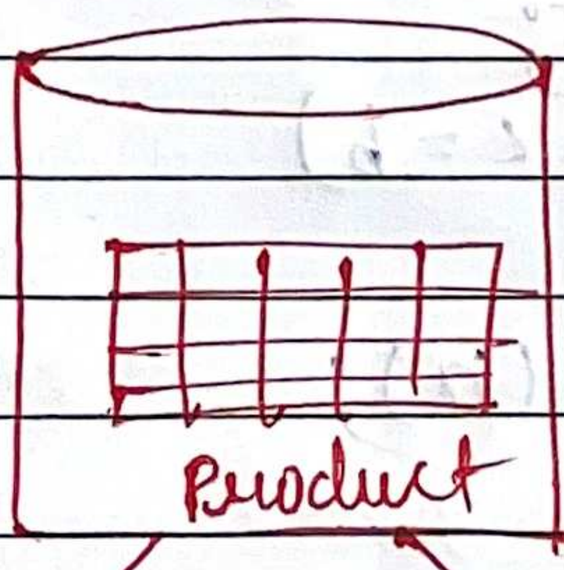
M T W T F S S

COMPASS

Date :

# Difference b/w FOR loop & WHILE loop

| | FOR | WHILE |
|---|---|---|
| 1. | No. of Iteration is knows | 1. No. of iteration is not known |
| 2. | In for loop declaration we can provide inititization and operation | 2. In while loop declation we cannot provide initilization and operation |
| 3. | Syntax<br>for(initialization ; condition; operation)<br>{<br>// for body<br>} | 3 Syntax<br>while (condition)<br>{<br>// while body<br>} |




```
for(int row=1 ; row<=100; row++)
{
/*
Java logic to fetch
data from product table
*/
}
```

```
while (row!=
Empty)
{
/*
*/
}
```

M T W T F S S

COMPASS

Date :

data.xlsx

```
for (int x=1; x<4; x++)
{ /*
    read data from
        file
*/
}
```

```
while (data != Empty)
{ /*
    read data
    from file
*/
}
```

```
#   int a = 1; // 2, 3, 4, 5, 6
    int b = 5;
    while (a <= b)
    {
        S.O.pln(a);
        a++;
    }
```

smaller
1 <= 2 or 3 <= 4
higher 5

output
1
2
3
4
5

```
#   int a = 5 // 4, 3, 2, 1, 0
    int b = 1;
    while (a >= b)
    {
        S.O.pln(a)
        a--;
    }
```

higher
5 > 4 or 3 >= 2 >= 1
smaller

output
5
4
3
2
1

M T W T F S S

COMPASS

Date :

# int a = 1; // 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
int b = 10;

Output

```
while (a <= b)
{                                   2
                                    4
    if (a%2 == 0)                   6
    {                               8
        S.O.P(a);                   10
    }
    a++
}
```

# int a = 1; // 2, 3, 5, 6.
int b = 5;
int sum = 0; // 1, 3, 6, 10, 15

```
while (a <= b)
{
    sum = sum + a;
    a++;
}
S.O.P(sum);
```

# Sum of even no. from 1 to 10

M T W T F S S

# DO-WHILE

COMPASS
Date : 08/04/25

# Syntax ⇒

```
do
{
  // body of do while
} while(Condition);
```

#
```
int a = 1;
do
{
  S.O.println("Ispider")
    a++;
} while (a<=5);
```

a = 1 Qspider
a = 2 ✓ Qspider
a = 3 ✓ Qspider
a = 4 ✓ Qspider
a = 5 ✓ Qspider.
a = 6 ✗

#
```
int x = 5
do
{
  S.O.println("Java")
    x--;
}
while (x>0);
```

x = 5 ✓ Java
x = 4 ✓ Java
x = 3 ✓ Java
x = 2 ✓ Java
x = 1 ✓ Java
x = 0 ✗

*

i <= num/2

| num | divisior | Perfect | count |
|---|---|---|---|
| 10 | 1, 2, 5 | | 3 |
| 15 | 1, 3, 5 | | 3 |
| 12 | 1, 2, 3, 4, 6 | | 5 |
| 6 | 1, 2, 3 | (1+2+3) = 6 | 3 |
| 28 | 1, 2, 4, 7, 14 | (1+2+4+7+14) = 28 | 5 |

M T W T F S S

COMPASS

Date :

| WHILE | DO-WHILE |
|---|---|
| → Entry checked loop | → Exit checked loop |
| → Executes only when condition is true | → Execute once irrespective of condition |
| → condition checked before loop execution | → Condition checked after loop execution |
| → Syntax – | → syntax – |

```
while(condition)
{
// body of while
}
```

```
do
{
// body of do while
} while (condition);
```

#
```
int num = 10;
int a = 1;
while (a <= num/2);
{
if (num % a == 0)
{
S.O.pln(a);
}
a++;
}
```

a = 1 ✓ 10 % 1 == 0 ✓
a = 2 ✓ 10 % 2 == 0
a = 3 ✓ 10 % 3 == 0 ✗
a = 4 ✗ 10 % 4 == 0 ✗
a = 5 ✓ 10 % 5 == 0 ✓
a = 6 ✗ ~~10 % 6 == 0 ✗~~
~~a = 7 ✗~~

1
2
5

#
```
int num = 12;
int a = 1;
while (a <= num/2);
{
if (num % a == 0)
{
S.O.pln(a);
}
a++;
}
```

a = 1 ✓ 12 % 1 == 0 ✓
a = 2 ✓ 12 % 2 == 0 ✓
a = 3 ✓ 12 % 3 == 0 ✓
a = 4 ✓ 12 % 4 == 0 ✓
a = 5 ✓ 12 % 5 == 0 ✗
a = 6 ✓ 12 % 6 == 0 ✓
a = 7 ✗

1, 2, 3, 4, 6

## #1

```
int num = 12;
int q = 1; // 2, 3, 4, 5, 6, 7
int sum = 0; // 1, 3, 6, 10, 16

while (q <= num/2)
{
  if (num % q == 0)
  {
    sum = sum + q;
  }
  q++;
}
System.o.println("sum of Divisors:" + sum);
}
}
```

12 % 1 == 0 ✓
12 % 2 == 0 ✓
12 % 3 == 0 ✓
12 % 4 == 0 ✓
12 % 5 == 0 ✗
12 % 6 == 0 ✓

## #2

```
int num = 28;
int q = 1 // 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
int count = 0; // 1, 2, 3, 4, 5

while (q <= num/2)
{
  if (num % q == 0)
  {
    count++;
  }
  q++;
}
S.o.println("Divisor Count: " + count);
```

M T W T F S S

Date : 09/04/25

## * IN-FINITE LOOP

A for loop or a while loop is which condition is always true is called as a infinit ∞ loop.

It is the programmers responsibility to ensure there are no infinte loop is a program.

### 1° IF-Statement

→ It is a branching statement that can be used by the programmer to create decision blocks inside a java program

→ Java provides the following versions of If-Statement.

(a) Simple If statement
(b) If-else statement
(c) If-else-If statement
(d) Nested If statement

### 2. SWITCH-STATEMENT

→ It is a selection statement that can be used by the programmer to make a selection for from a list of predefined options.

→ Switch Statement can be used by the programmer for MCQ Scenarios.

M T W T F S S

COMPASS

Date :

3. FOR LOOP :

It is a loop control statement that can be used by the programmer when no of iterations is known.

4. WHILE LOOP :

It is a loop control statement that can be used by the programmer when no of iterations is unknown.

5. DO - WHILE LOOP :-

It is variant of the while loop and can be used by the programmer when there is a need for minimum 1 - iteration before exit.

M T W T F S S

COMPASS

Date :

# METHODS

→ A method is a named block of code that is reusable in nature.

→ Method are also known as functions.

→ The syntax for creating a method consist of two parts

1. Method Declaration
2. Method Definition

Syntax:

<access_Modifier> <modifier> returntype } Method Declaration
methodname (<argument>)
{ }
→ Method Definition

<Optional>

```
class Program1
{
  static void test()
  {
    S.O.Pln("Executing play() : ...");
  }
  public static void main(String[] args)
  {
    S.O.Pln("Program Starts")
    test();
```

M T W T F S S

COMPASS

Date :

```
        S.O.pln("Program Ends");
    }
}
```

# class Program2
```
{
    public static void main(String[] args)
    {
        S.O.pln("Program Start");
        test();
        test();
        S.O.pln("Program End");
    }
    Static void test()
    {
        S.O.pln("          ");
    }
}
```

* A method should always be declared directly in the class body.
* A method gets executed only when it is called.
* We can call the same method 'n' number of times.
* Depending on ^our ~~the~~ requirement we can declare multiple methods in our class.
* method provieds modularity in the program making it easy to maintain modify and understand.

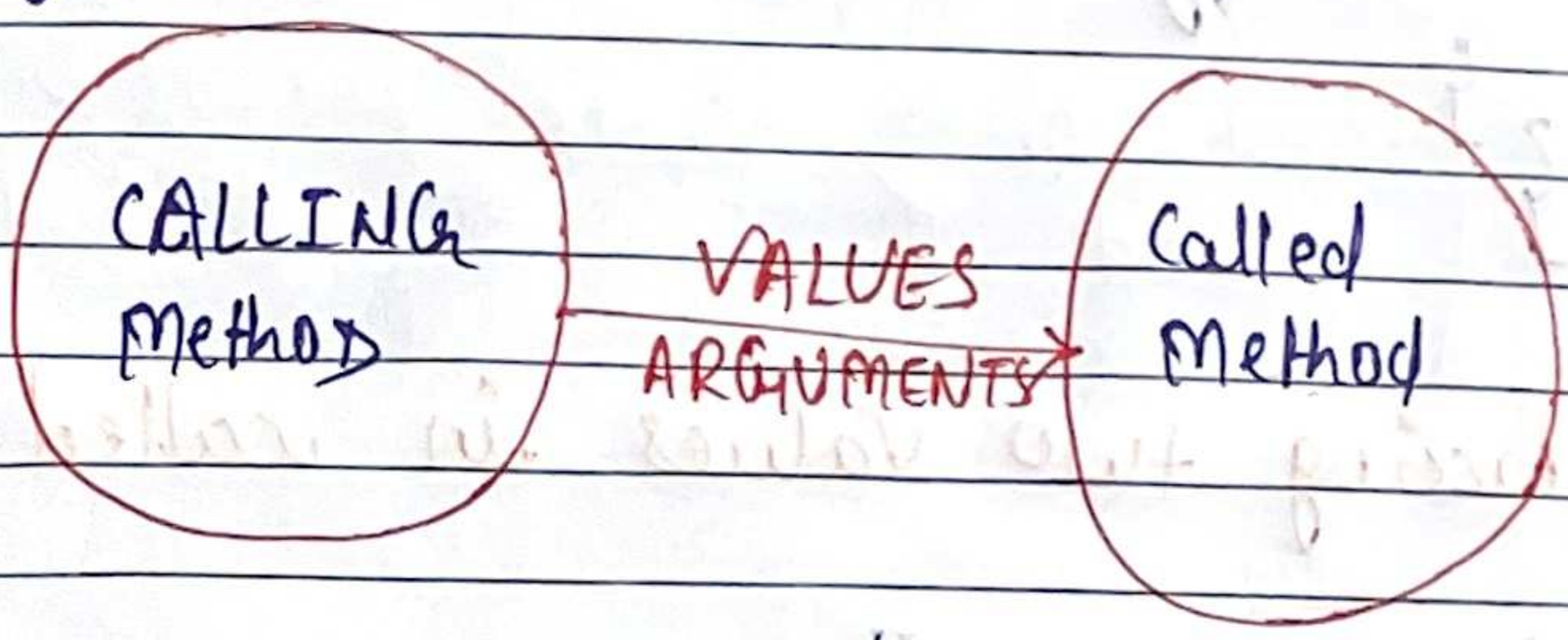
M T W T F S S

COMPASS

Date : 11/04/25

## * Passing Values:

→ It is the consept of sending value from calling method to called method.




# class programs

```
{
    sutt static void test(int a)        ← formal argument
    {
        S.O.pln("value: "+a);
    }

Public static void main(String [] args)   ← calling method
    {
        test(10);                         ← Actual Argument
    }                                     ← Called Method
```

# Class Program7

```
{
Static void verify (int otp)
 { if (otp==1234)
  {S.O.pln ("Valid otp");
   }
  else
   {S.O.pln ("Invalid otp");
   }
 }
```

M T W T F S S

COMPASS

Date :

```
P. S. V main (string [] args)
{
    verify (2845);
    verify (12345);
}
```

# Passing two values in called method

```
class Program9
{
    static void addition (int a, int b)
    {
        int sum = a+b;
        S.O.pln(a+"+" +b+ "=" + sum);
    }
    public static void main (String [] args)
    {
        addition (12, 13); (Passing two values)
        addition (100, 200);
    }
}
```

# Passing different value (different data type)

```
class Program11
{
    static void display (String name,
        int yop, double cgpa, char gender)
    { S.O.ptln ("Name :" + name);
```

M T W T F S S

COMPASS

Date :

```
        S.O.pln("Yop: " +yop);
        S.O.pln("CGPA: " +cgpa);
        S.O.pln("Gender:" +gender);
    }
    public static void main(String[] args)
    {
        display("shrastii Pawar", 2024, 8.42, 'F');
    }
}
```

NOTES

* WE can pass values with the help of argument.

→ Actual arguments : This are the argument values that we provide when we call the method.

→ Formal Arguments :- This are the veriables created in method declaration to store the actual argument

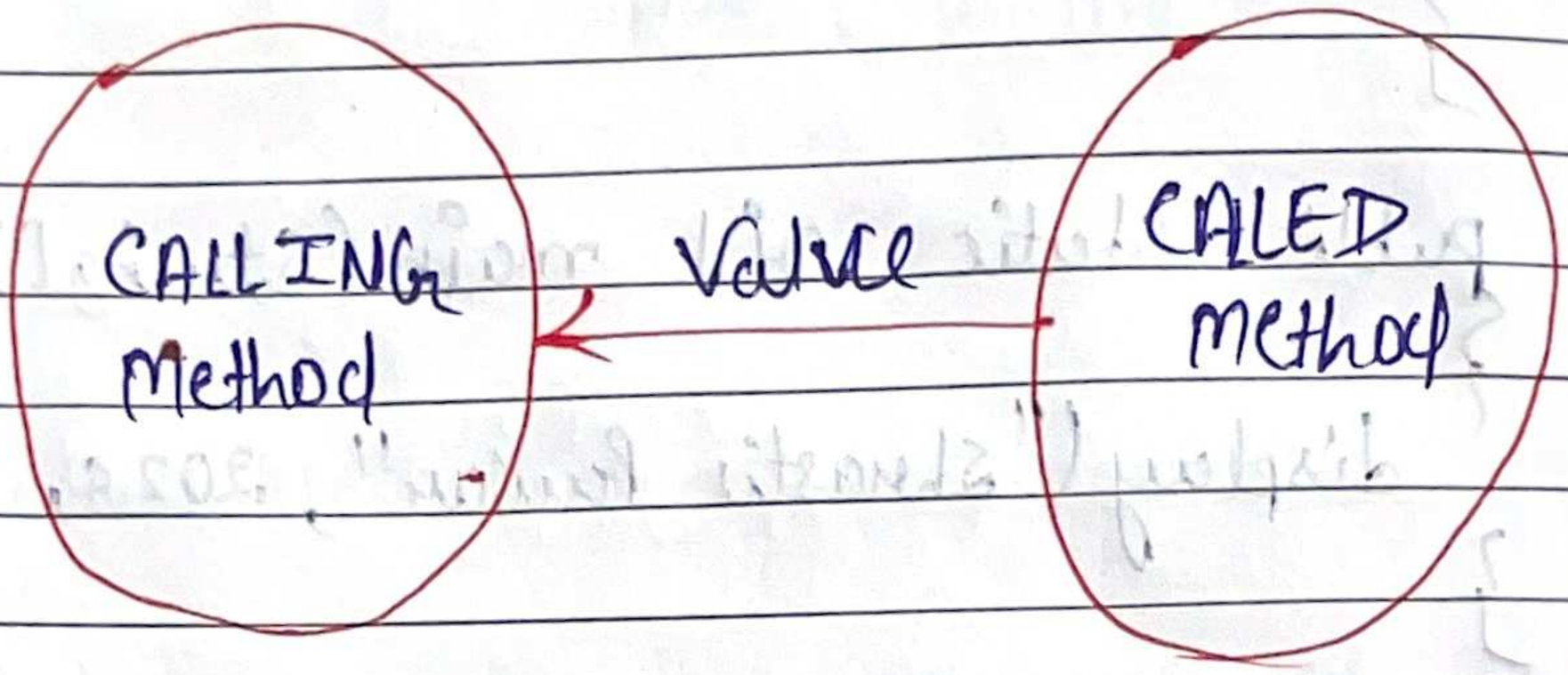
* If ner the programer can pass multiple values seperated by (,).
* If is the programmers responsiblity to provide arguments of the currect type, length and sequence

M T W T F S S

Date : 12/04/25

## * RETURNING VALUE :

→ It is a concept of sending a value from called method to calling method




→ To return a value a programmer below three steps:

1. Declare return type based on the value of to be returned
2. Return a compatible value using return keyword.
3. capture the returned value inside the calling method.

```
# class Program12
  {
     static int test()
     {
        return 25;
     }

  public static void main(String[] args)
  {
     int val = test();
     System.out.println("Returned value :" +
                        val);
     System.out.println("Returned value :" +
                        test());
  }
}
```

M T W T F S S

COMPASS

Date :

# static char Play()
{
return 'J';
}

Static boolean help().
{
return true;
}

Public static main()
{ int val = play()
S.O.pln ("value :" +ch);
S.O.pln ("value :" +help);
}
}

# Passing & return value both

# class Program1
{
static boolean verify (int val)
{
if (val % 2 == 0)
{
return true;
}
else
{
return false;
}
}

COMPASS

M T W T F S S

Date :

```
public static void main (String [] arg
{
    s.o.pln ("Even Number:" + verify((18));
}
}
```

# class Program 2
```
{
  static int Addition (int a, int b)
  {
    int sum = a+b;
    return sum;
  }

public ------
  {
    s.o.pln ("sum:" + addition (10,20));
  }
}
```

# Program for Prime Number:

```
class Program 4
{
  static boolean findPrimeNumber (int num)
  {
    int a=1;
    int count =0;
    while (a<=num/2)
    {
      if
```

M T W T F S S

COMPASS

Date :

ex

```
class Scenario 1
{
    static void test()
    {
        disp()
    }
    static void disp()
    {test();
}}
```

2. Scenario : 2

In scenario 2 the called method will call back to It self.

ex

```
class Scenario 2
{
    static void help()
    {
        help();
    }
}
```

" called It self

Note

During recursion It is the programmer responsibility to ensure there is a exit point to the recursion otherwise it will cause StackOverflowError

M T W T F S S

Date :

```
class Program1
{
    static void display (int a)
    {
        System.out.println (a);

        if (a<5)
        {
            a++
            display(a);
        }
    }
    public static void main(String[] args)
    {
        display (1);
    }
}
```